

An Array–Based Implementation

```
#pragma once

const int MAXLIST = 10;
template<class T>
class ListArray{
private:
    int size;
    T *items;
    int translate(int index) const;
public:
    ListArray(int n);
    ListArray(ListArray &);
    ListArray();
    ~ListArray();
    bool isEmpty();
    int getLength() const;
    void insert(int index, T newItem, bool &success);
    void remove(int index, bool & success);
    void retrieve(int index, T &dataItem, bool sucess) const;
    void print() const;
};

//end ListArray
```

An Array –Based Implementation

```
template<class T>
ListArray<T>::ListArray(int n):size(0){
    n = (n>0 ? n:MAXLIST);
    items = new T[n];
}//end constructor
```

```
template<class T>
ListArray<T>::ListArray(ListArray &tempListArray){
//implement
}//end constructor
```

```
template<class T>
ListArray<T>::ListArray():size(0)
{
    items = new T[MAXLIST];
}//end default constructor
```

```
template<class T>
ListArray<T>::~ListArray(){
    delete [] items;
}//end destructor
```

An Array –Based Implementation

```
template<class T>
bool ListArray<T>::isEmpty(){
    return bool(size ==0);
} //end isEmpty

template<class T>
int ListArray<T>::getLength() const{
    return size;
}//end getLength

template<class T>
int ListArray<T>:: translate(int index) const
{
    return (index - 1);
}//end translate

template<class T>
void ListArray<T>::insert(int index, T newItem, bool &success){
    success = bool ((index >=1)&&(index<=size+1) && (size<MAXLIST));
    if(success){
        for(int pos=size;pos >=index;--pos)
            items[translate(pos+1)]=items[translate(pos)];

        items[translate(index)]=newItem;
        ++size;
    }//end if
}//end insert
```

An Array –Based Implementation

```
template<class T>
void ListArray<T>::remove(int index, bool & success){
    success = bool((index>=1) && (index<=size));
    if(success){
        for(int fromPosition = index+1; fromPosition<=size; ++fromPosition)
            items[translate(fromPosition-1)]=items[translate(fromPosition)];
        --size;
    } //end if
} //end remove

template<class T>
void ListArray<T>::retrieve(int index, T &dataItem, bool success) const{
    success= bool((index >=1) && (index <= size));
    if(success)
        dataItem=items[translate(index)];
} //end retrieve

template<class T>
void ListArray<T>::print() const{
    for(int i=0;i<size;i++)
        cout<<items[i]<<",";
    cout<<endl;
} //end print
```

An Array –Based Implementation

```
#include<iostream>
using namespace::std;
#include "ListArray.h"

int main()
{
    ListArray<int> aList(6);
    bool success;
    int dato;

    aList.insert(1,20,success);
    if(success) aList.print();
    else cout<<"Index out of range\n";
    aList.insert(2,23, success);
    if(success) aList.print();
    else cout<<"Index out of range\n";
    aList.insert(11,34,success);
    if(success) aList.print();
    else cout<<"Index out of range\n";
    aList.insert(3,67,success);
    aList.print();
    aList.insert(3,78,success);
    aList.print();
    aList.insert(3,34,success);
    aList.print();
    cout<<"Array size:"<<aList.getLength()<<endl;

    aList.retrieve(2,dato,success);
    cout<<"Data:"<<dato<<endl;
    aList.remove(3,success);
    if(success) aList.print();
    else cout<<"Index out of range\n";
    cout<<"Array size:"<<aList.getLength()<<endl;
    return 0;
}//end main
```

20,
20,23,
Index out of range 20,23,67,
20,23,78,67,
20,23,34,78,67,
Array size:5
Dato:23
20,23,78,67,
Array size:4
Press any key to continue ..
.

- Implement :
 - 1) Problem #14 Chapter 11
Book: Starting out with C++ 6th Edition.
Must use the above structures.
Use the functions insert, retrieve, delete, option menu.
 - 2) Copy constructor